TAUCHGANG 02

Die Werkzeuge.

Slash-Commands, Subagents, Memory und das 3-Tool-Setup, mit dem ich arbeite.

Dauer: ca. 90 Min Niveau: Standard Teil des Kurses: Tauchgang

> **Video:** Der Modul-Trailer zu diesem Tauchgang ist Teil der Online-Lektion. Schau ihn dir an unter sebastian-sperber.de/tauchgang/lektion-02.

> ***Dauer:*** ***ca. 90 Minuten*** ***Voraussetzung:*** ***Tauchgang 01 ist durch. Claude Code läuft, dein Vault steht.*** ***Was du danach hast:*** ***Wiederverwendbare Slash-Commands, eigene Skills, persistentes Memory, und Ultra Plan für komplexe Aufgaben. Du hast einen Werkzeugkoffer, nicht nur eine Chat-App.***

Vor dem Tauchgang: Warum Werkzeuge, nicht Chats?

Chat ist flüchtig. Du erklärst, Claude macht, du vergisst. Nächste Woche erklärst du wieder.

Werkzeuge sind anders. Ein Skill, den du einmal schreibst, gilt für immer. Ein Slash-Command, den du einmal anlegst, startet einen kompletten Workflow. Ein Subagent, den du einmal installierst, ist ab dann immer da.

Dieser Tauchgang baut dir vier Kategorien von Werkzeugen: 1. **Globale Plugins** — die Claude-Code-Community hat viel gebaut, wir nutzen das Beste 2. **Eigene Skills** — Regeln, die Claude für deine Themen beherrscht 3. **Slash-Commands** — fertige Workflows, die du mit `/befehl` startest 4. **Memory & Subagents** — persistentes Gedächtnis und spezialisierte Agenten

Lektion 2.1 — Globale Plugins installieren

Ziel: Drei Plugins, die Claude deutlich besser machen. Einmal installiert, überall verfügbar.

Plugin 1 · Superpowers — das größte Claude-Code-Plugin

Was es macht: Zwingt Claude, erst nachzudenken bevor es codet. 14+ Skills drin: Brainstorming vor Feature, Test-Driven Development, systematisches Debugging, automatische Code-Reviews.

Installation:

```
/plugin install superpowers@claude-plugins-official
```

Slash-Commands, die ab jetzt verfügbar sind:

- `/superpowers:brainstorming` — Feature vor dem Code durchdenken
- `/superpowers:writing-plans` — strukturierter Plan mit Milestones
- `/superpowers:systematic-debugging` — Bug isolieren, nicht raten
- `/superpowers:executing-plans` — Plan Schritt für Schritt abarbeiten

Plugin 2 · Skill Creator — der Meta-Skill

Was es macht: Ein Skill, der andere Skills erstellt. Vier Modi: Create, Eval, Improve, Benchmark.

```
/plugin install skill-creator@claude-plugins-official
```

Slash-Command:

- `/skill-creator:skill-creator` — startet den interaktiven Skill-Erstellungsprozess

Du beschreibst einen Workflow, Skill Creator baut die Skill-Datei für dich.

Plugin 3 · Frontend Design — gegen den generischen AI-Look

Was es macht: Gibt Claude ein Design-System BEVOR es Code schreibt. 67 verschiedene Design-Stile (Brutalist, Luxury, Retro-Futuristic etc.), distinktive Typografie, durchdachte Farbpaletten.

```
/plugin install frontend-design@claude-plugins-official
```

Slash-Command:

- `/frontend-design:frontend-design` — startet Design-Phase vor Code

Aktiviert sich automatisch bei Frontend-Aufgaben. Resultat: Deine Websites sehen nicht mehr nach AI aus.

Nach der Installation einmal Claude Code neu starten, damit die Plugins laden.

Plugin prüfen

```
claude plugin list
```

Alle drei sollten drin stehen. Wenn nicht — ein neues Terminal öffnen und nochmal probieren.

Lektion 2.2 — Deine eigenen Skills bauen

Ziel: Skills, die deine Agentur-Workflows kennen. Die werden automatisch geladen, wenn du sie brauchst.

Das Skill-Prinzip in 30 Sekunden

Ein Skill ist eine Markdown-Datei in `.claude/skills/` (pro Projekt) oder `~/.claude/skills/` (global). Oben ein YAML-Frontmatter, unten Anweisungen für Claude.

Beispiel — `website-audit.md`:

```
---
name: website-audit
description: Durchläuft einen 30-Min-Website-Check. Nutze dies wenn "Website-Check" oder "Audit" e
---

Du machst einen 30-Min-Website-Check nach meinem System.

Struktur:
1. Technik: Ladezeit, Mobile, SSL, Uptime (PageSpeed Insights, wave.webaim.org)
2. SEO: Title, Description, H1, Schema (Sichtcheck + Rank Math wenn WordPress)
3. UX: Navigation, CTA-Klarheit, Kontaktwege (3-Klick-Regel)
4. Content: Tonalität, Zielgruppen-Fit, Aktualität
5. Conversion: Welcher Pfad zur Anfrage? Reibung? Vertrauens-Signale?

Format-Output:

- 5 Ampel-Sektionen (rot/gelb/grün) mit je 2-3 konkreten To-Dos
- Schluss: Top-3 Hebel (was hat den größten Effekt bei kleinstem Aufwand)
- Am Ende: Angebot-Platzhalter mit meinem Stundensatz und Aufwand-Schätzung
```

Speichern → ab jetzt greift der Skill automatisch, sobald du in Claude sagst: „Mach einen Website-Check für [url]“.

Skill-Ideen für Agenturen (alle bewährt bei mir)

SKILL	WAS ER MACHT	WANN ER GREIFT
`website-audit`	30-Min-Check mit Ampel-Output	„Website-Check”, „Audit”
`angebot-schreiben`	Angebot nach § 19 UStG, mit Stundensatz-Logik	„Angebot”, „Offerte”
`rechnung-erstellen`	Rechnung ohne MwSt., korrekte Nummerierung	„Rechnung”
`social-post-agentur`	LinkedIn/IG-Post nach deinem Tone of Voice	„Social-Post”
`content-ideas`	Tägliche Video-Ideen basierend auf Trends	„Content-Idee”
`meeting-protokoll`	Nach Call: Zusammenfassung + nächste Schritte	„Meeting-Notes”

Skill erstellen mit dem Skill Creator

Am schnellsten: Skill Creator nutzen (Plugin aus Lektion 2.1).

```
/skill-creator:skill-creator
```

Beispiel-Dialog:

- SC: „Was soll der Skill können?”
- Du: „Einen Website-Check nach meinem 5-Punkte-Schema durchführen.”
- SC: „Welche Trigger-Wörter sollen ihn auslösen?”
- Du: „Website-Check, Audit, Analyse”
- SC: „Soll ich Testfälle schreiben?”
- Du: „Ja, mit 3 Beispiel-URLs.”

Skill Creator baut die Datei, testet sie, speichert unter `~/.claude/skills/website-audit.md`. Fertig.

Lektion 2.3 — Slash-Commands: Workflows mit einem Befehl

Ziel: Komplette Abläufe, die du mit einem `/befehl` startest. Keine 30-Sekunden-Kontext-Erklärung mehr.

Struktur

Slash-Commands liegen in `~/.claude/commands/` (global) oder `.claude/commands/` (pro Projekt).

Beispiel — `~/.claude/commands/neukunde.md`:

```
---
description: Legt einen neuen Kunden im Vault an und öffnet Angebot-Vorlage
---

Du legst einen neuen Kunden an.

Schritte:
1. Lies den Namen aus $ARGUMENTS
2. Erstelle eine neue ID: nimm die höchste bestehende ID in 04-crm/kunden/ + 1 (3-stellig)
3. Lege an: 04-crm/kunden/<id>-<name>.md mit meinem Standard-Template
4. Frage nach: Kontakt-Daten (Mail, Tel), Website, geplantes Projekt
5. Wenn alle Daten da: Erstelle 02-proposals/<YYYY-MM-DD>-<id>-<projekt>.md aus 07-templates/angeb
6. Fülle Platzhalter: Datum (heute), Kunde, Projektbeschreibung, Stundensatz (82,50 € Neukunde)
7. Zeig mir beide Dateien zum Review
```

Nutzung:

```
/neukunde Zahnarzt Röthenbach
```

Claude legt die Client-Hub-Page an, fragt die Daten ab, erstellt das Angebot. 20 Minuten Arbeit, die du früher manuell gemacht hast — jetzt 2 Minuten Review.

Weitere Slash-Command-Ideen

- `/call-prep <kunden-id>` — liest CRM-Datei + letzte Daily-Logs, spuckt Call-Briefing aus
- `/monatsrechnung <kunden-id>` — summiert Stunden aus Timetracking, erstellt Rechnung
- `/content-plan` — liest 03-marketing/, schlägt 7 Posts für nächste Woche vor
- `/website-check <url>` — startet den Website-Audit-Skill mit einer URL

Lektion 2.4 — Memory & Subagents

Ziel: Claude erinnert sich über Sessions hinweg. Spezialisierte Subagents arbeiten parallel.

claude-mem — persistentes Gedächtnis

Das Problem: Claude vergisst am Ende einer Session alles, was nicht in CLAUDE.md steht.

Die Lösung: **claude-mem** — ein Hook, der jede Session mitschreibt, komprimiert, in künftige Sessions injiziert.

Installation (einmal, global):

```
npx claude-mem install
```

Verifizieren:

```
npx claude-mem status
```

Ausgabe sollte sein: `claude-mem active — memory layer installed`.

Pro-Tipp: Nach der Installation eine Session starten, ein paar Aufgaben machen, Session schließen. Neue Session öffnen — Claude „erinnert" sich an den vorherigen Stand. Ohne dass du was erklärst.

Zwischen Projekten trennen:

```
/clear-memory
```

Das nutzt du, wenn du an einem neuen Kundenprojekt anfängst und nicht willst, dass der Kontext von Projekt A reinblutet.

Subagents — spezialisierte Agenten für spezifische Aufgaben

Statt einem Generalisten für alles: spezialisierte Agenten. 100+ gibt es im Awesome-Repo.

Repo klonen (einmal):

```
git clone https://github.com/VoltAgent/awesome-claude-code-subagents /tmp/subagents
```

Die gewünschten kopieren. Für Agentur-Arbeit sinnvoll:

```
mkdir -p ~/.claude/agents
cp /tmp/subagents/agents/frontend-engineer.md ~/.claude/agents/
cp /tmp/subagents/agents/security-auditor.md ~/.claude/agents/
cp /tmp/subagents/agents/test-writer.md ~/.claude/agents/
cp /tmp/subagents/agents/technical-writer.md ~/.claude/agents/
```

Aufrufen:

```
/frontend-engineer Erstelle die Login-Seite für mein React-Projekt
/security-auditor Prüfe die Authentifizierung auf Lücken
/test-writer Schreib Unit-Tests für die Auth-Funktionen
```

Pro-Tipp: Mehrere Agents nacheinander auf das gleiche Feature ansetzen — erst Frontend-Engineer für die UI, dann Security-Auditor für die Prüfung. Das entspricht einem echten Code-Review-Prozess. Du bist quasi Teamlead.

Lektion 2.5 — Ultra Plan für komplexe Projekte

Ziel: Bei großen Aufgaben planst du in der Cloud, mit 3 parallelen Research-Agents, und implementierst dann lokal.

Voraussetzungen

Ultra Plan braucht drei Dinge: 1. Claude Code Pro, Max, Team oder Enterprise Plan 2. Dein Projekt muss ein GitHub-Repository sein 3. GitHub muss mit Claude Code verbunden sein (einmalig)

GitHub verbinden

Im Terminal:

```
/web-setup
```

Folgen den Anweisungen, GitHub autorisieren. Ab dann ist dein Account verbunden.

Ultra Plan starten

Drei Wege, alle gültig:

Weg 1 — Slash Command (empfohlen):

```
/ultraplan Baue mir ein Dashboard mit Umsatzübersicht, Kundenstatistiken und Support-Tickets. Nutz
```

Weg 2 — Keyword im Prompt: Einfach „ultraplan" irgendwo im Text erwähnen → Bestätigungsdialog erscheint.

Weg 3 — Aus bestehendem Plan: Wenn Claude einen lokalen Plan fertig hat → „Refine with Ultraplan on the web" wählen.

Plan im Browser reviewen

Nach dem Start bekommst du einen Link. Im Browser siehst du den strukturierten Plan mit Diagrammen und Sektionen:

- **Inline Comments** — Text markieren und Kommentare auf einzelne Abschnitte hinterlassen
- **Emoji Reactions** — Zustimmung oder Bedenken auf bestimmte Stellen signalisieren
- **Mehrfach iterieren** — Änderungen anfordern, Claude überarbeitet den Plan

Dein Terminal bleibt währenddessen frei — du kannst dort weiterarbeiten.

Plan ausführen — Option B (empfohlen)

Wenn der Plan fertig ist, wähle **„Approve plan and teleport back to terminal"**.

Der Plan wird ins Terminal geschickt. Du wählst: „Implement here" oder „Start new session". Claude Code baut alles lokal — mit deinen Skills, MCPs, Tools.

Warum lokal und nicht in der Cloud? Lokal hast du volle Kontrolle, alle deine Skills, dein Memory, deine Fonts, deine Datenbanken. In der Cloud sind nur Claude Code und das Repo.

Wann Ultra Plan nutzen

- **Ja:** Komplexe Projekte mit vielen Dateien (>20 Files, neue Architektur)
- **Ja:** Wenn du ein besseres Review-Interface als das Terminal willst
- **Ja:** Wenn du dein Terminal während der Planung freihalten willst
- **Nein:** Für einfache Änderungen — der Overhead (Repo klonen, Cloud-Session) lohnt nicht

---

Lektion 2.6 — Das 3-Tool-Setup für Shippbare Projekte

Drei Werkzeuge, die zusammen dafür sorgen, dass Projekte wirklich fertig werden — nicht halb in der Schublade liegen.

Tool 1 · claude-mem (hatten wir schon)

Context Loss gelöst. Session-übergreifendes Memory.

Tool 2 · GSD (Get Shit Done) — das Context-Engineering-System

GitHub: github.com/gsd-build/get-shit-done

GSD beschreibt was du bauen willst → GSD erstellt einen strukturierten Plan mit Quality Gates und arbeitet ihn Schritt für Schritt ab. Eingebaut: Schema-Drift-Detection, Security-Enforcement, Scope-Reduction-Detection.

Installation global:

```
npx get-shit-done-cc --claude --global
```

Oder für ein einzelnes Projekt:

```
npx get-shit-done-cc --claude --local
```

Ersten Plan erstellen — Claude Code starten, dann:

```
/gsd Ich will eine Web-App bauen die [dein Projekt beschreiben].
Erstelle einen vollständigen Plan mit Milestones.
```

GSD gibt einen strukturierten Plan zurück mit nummerierten Schritten, Abhängigkeiten und Definition of Done für jeden Schritt.

Tool 3 · Kombination

Kompletter Workflow, wenn alle drei zusammenspielen:

```
# Session starten — claude-mem lädt automatisch den Kontext der letzten Session

# Plan erstellen
/gsd Ich baue eine SaaS-App mit User-Auth, Dashboard und Stripe-Payment.
Erstelle den Plan.

# Spezialisierte Agents für jeden Teil
/backend-engineer Implementiere Schritt 1 aus dem GSD-Plan: User-Auth mit JWT
/frontend-engineer Implementiere Schritt 2: Login-UI
/test-writer Schreibe Tests für Auth und UI

# Session schließen, morgen weitermachen
# claude-mem injiziert den Stand automatisch — Claude weiß wo du aufgehört hast
```

Pro-Tipp: Den GSD-Plan als `PLAN.md` im Projektordner speichern. So können alle Agents jederzeit nachlesen, was der aktuelle Milestone ist.

Häufige Fehler

- `npx claude-mem install` schlägt fehl: Node.js nicht installiert → nodejs.org, Installer, neu starten
- `/gsd` nicht verfügbar nach Installation: Falscher Install-Pfad → `npx get-shit-done-cc --claude --global` nochmal ausführen, neues Terminal
- Subagent-Datei nicht erkannt: Datei liegt nicht unter `~/.claude/agents/` → `ls ~/.claude/agents/` prüfen, Datei muss `.md` -Endung haben
- claude-mem zeigt alten Context aus falschem Projekt: Memory ist global, nicht projekt-spezifisch → zu Beginn neuer Projekte `/clear-memory` aufrufen
- GSD erstellt Plan aber Agent weicht ab: Scope-Drift → `/gsd status` aufrufen um den aktuellen Plan-Stand zurückzubekommen

---

Was nach Tauchgang 02 läuft

- [ ] Superpowers, Skill Creator, Frontend Design installiert
- [ ] Mindestens 2 eigene Skills geschrieben (website-audit + angebot-schreiben)
- [ ] Mindestens 2 Slash-Commands angelegt (/neukunde + /call-prep)
- [ ] claude-mem läuft, Memory ist aktiv
- [ ] 3 Subagents installiert (frontend-engineer, security-auditor, technical-writer)
- [ ] Optional: Ultra Plan einmal ausprobiert
- [ ] Optional: GSD installiert und ersten Plan erstellt

Links aus diesem Tauchgang

- Plugin-Verzeichnis: via `/plugin` im Terminal
- Superpowers: Claude Plugin Official Repo
- Skill Creator: Claude Plugin Official Repo
- Frontend Design: Claude Plugin Official Repo
- claude-mem: github.com/thedotmack/claude-mem
- GSD: github.com/gsd-build/get-shit-done
- Awesome Subagents: github.com/VoltAgent/awesome-claude-code-subagents

- Ultra Plan Docs: code.claude.com/docs/en/ultraplan
- Web Setup Docs: code.claude.com/docs/en/claude-code-on-the-web

---

Nächster Tauchgang: 03 — Die Strömung. Remotion, Nano Banana für Design-Assets, WordPress-Websites in unter einem Tag. Wo der Content wirklich aus deiner Hand fließt.

BONUS

Deine Werkzeugkiste

Acht PDFs zum Download. Unabhängig von diesem Modul direkt einsetzbar.

01
5 KI-Prompts für Agenturen
PDF · SOFORT EINSETZBAR

02
Content-Brief Baukasten
PDF · TEMPLATE

03
Angebots-Vorlage
PDF · § 19 USTG-READY

04
Meeting-Protokoll
PDF · STRUKTUR-PROMPT

05
Mein KI-Stack
PDF · TOOL-AUSWAHL

06
30-Tage-Plan
PDF · EINFÜHRUNGS-PLAN

07
Style-Guide Baukasten
PDF · MARKENTON-TEMPLATE

08
KI-Audit Fragebogen
PDF · FÜR KUNDENGESPRÄCHE

Weiter tauchen?

Alle vier Module sind Teil des Kurses „Tauchgang".

Zur Kursübersicht

Kurz-Anleitungen zu diesem Tauchgang

Vier 45–Sekunden-Videos — jeweils ein konkreter Schritt zum Nachmachen.

01 **Slash-Command bauen**
Wiederverwendbarer Prompt

02 **Skill schreiben**
Spezialisierte Markdown-Rolle

03 **Subagent einsetzen**
Parallele Tasks delegieren

04 **Eigenes 10-Prompt-Kit**
Agentur-Kern-Prompts

Mediengestalter · Art Director · Röthenbach a.d. Pegnitz
Kleinunternehmer nach § 19 UStG · keine Mehrwertsteuer